METHOD AND SYSTEM FOR MANAGING DATABASE SQL STATEMENTS IN WEB BASED AND CLIENT/SERVER APPLICATIONS

CROSS-REFERENCE TO RELATED APPLICATION

This application claims the benefit of Provisional Patent Application No. 60/452,643, filed March 6, 2003.

TECHNICAL FIELD

The present invention is related to application development and embedded relational database calls within application programs, and, in particular, to a method and system for providing database-independent and embedded-SQL/stored-procedure-independent calls for application development that can be easily toggled to function as interfaces to various different DBMSs and to function either as embedded database calls or as stored procedure calls.

BACKGROUND OF THE INVENTION

Typical problems associated with developing applications that interface to relational DBMSs include: the disconnect between a fluid design-time application environment vs. the "locked-down" run-time application environment; applications that break when database or systems architecture's change; and non-modular content and form components that have intertwined SQL statements and application code.

All of these problems are exacerbated by the increased use of SQL as a point for integration between applications, and as a source of business object rules. Applications continue to migrate towards database centric solutions, where more logic is stored with the database content rather than within the application code. Unfortunately, many developers embed much of the SQL used to access these business data and rules into the application code itself, negating some of the benefits.

One approach is to make a DBMS Stored Procedure call from the application code rather than building an SQL statement in the application code then calling the DBMS engine. But working with Stored Procedures is more cumbersome than file based SQL especially with other third party tools like source control, report writers, template design tools and merge utilities. Even SQL Query development tools delivered with the DBMS software or provided by third parties work more simply with file based SQL statements rather than stored procedures. Ultimately however, SQL wants to run as a stored procedure within the database for maximum speed and efficiency. This is the "disconnect" between design-time and run-time SQL development. Development tools are geared around building SQL statements in a file based architecture, and production databases are built around SQL statements coded within the database server.

In some cases application code is modified, typically at the last minute, to change from embedded SQL to embedded SP calls. But this means changing the code back to embedded SQL in order to easily add major features or it means modifying the stored procedure directly to correct SQL mistakes. When SQL code is done in stored procedure it becomes much more difficult to port to other database platforms as SP syntax and "language" definitions vary more greatly between DBMS then does the SQL syntax, normally written in the SQL-92 standard. Even many SQL statement syntaxes are DBMS particular and if these become embedded in the application code then the code is not portable between DBMS. One advantage to moving to stored procedures though is that applications have better separation between form or UI and application content making any application changes simpler. Either way there is no simple way to use the third party dev tools to test, debug and correct the SQL statements.

So if you keep the code as embedded SQL you have better DBMS platform portability but more difficulty making code changes, especially since many changes involve just the content logic. Every time you change application code you typically need to recompile, link and redistribute the code as well as thoroughly test the application. Additionally, when SQL statements are coded directly in code or as stored procedures the syntax used for parameters or variables typically need to express the data types or those parameters. So even if the SQL itself doesn't change the code may have to if the data types for variables used in the code are changed. This indicates a need for a product that supports multiple SQL coding techniques and has runtime substitution of tokens including determining their data types but also allows flexible design of SQL that can be easily put into stored procedures for performance reasons.

SUMMARY OF THE INVENTION

One embodiment of the present invention is the Tokenized SQL Architecure ("TSA"), composed of three important technologies: the tokenized SQL file or .SQT file, the ExpandTokenizedSQL function and the CSP utility. Enhancing the development advantages of TSA is any template based rendering engine and authoring environment.

The tokenized SQL file or .SQT file is a file format that contains a single SQL statement plus parameter and data type information (or tokens). This is a simple ASCII file using section and entry conventions similarl to a MS Windows based .INI file. The SQT file may include an [SQL] section and a [TOKENS] section. In TSA terms the variables or parameters used in any SQL are Tokens since they represent both a parameter, it's run-time or design-time value as well as data type information about the parameter. The embedded token syntax includes an @ symbol directly in front of the variable that you want expanded. This method was selected since it is identical to a stored procedure "named parameters" type syntax. Normally, the SQL is ready to run inside the target DBMS stored procedure environment as well as ready to run as dynamic SQL with the exception of specifying the parameter values.

The SQL statement can easily be cut and paste into query editors or other development tools with valid values to supply to the parameters included in the file. No application code programming syntax such as concatenation operators or quotes need to be modified in order to run the statement. If used with the AccessVia template authoring tool, no cut and paste operation is required. Important to making SQL easy to develop with is this tokenized SQL file or .SQT file.

The ExpandTokenizedSQL function can take a variety of SQL statement input sources, with the input type being declared by the source type parameter. Normally, in a development or design mode during the application development process, an SQT file name is provided with token names and values and the "File" source type. During production mode, this SQT file name will represent a stored procedure name and the appropriate parameter values specified along with the "Stored Procedure" source type. Other source types are also supported. See the function prototype in section 4.2 below.

The TSA includes an SQT2SP utility. This utility executable takes a folder of SQT files and generates all of the necessary stored procedure code for the database. This has been customized to use XML with DBMS specific XSL to easily support any new or existing DBMS. This utility reads an INI file that stores folder locations, a database connection string, and XSLT stylesheets to plug in at run time for the various database vendors. It will read in all of the SQT files in the source directory and process them. Optionally, it will read generate a new set of correct default values for entries in the [TOKENS] section. This insures that the default values stay in sync and up to date in the SQT file so that when a template author links to that SQT file it will run with current default values.

The TSA allows the code to be data type independent by pushing the SQL and data type specifications into the SQT file, without requiring difficult SQL parsing. The TSA specifies a matching table and column name reference for each parameter so that it's type can be easily tested. These entries are included in the [TOKENS] section. However, not all stored procedure parameters map directly to a table and column name so it is also an option to specify the specific data type. The table is opened to get the column info, which creates a performance hit, but in design mode this is fine, in production mode this will not be necessary as all of the data type information will have already been correctly generated for the stored procedure. The two purposes of these entries are to determine if quote characters need to be placed around the values for the appropriate parameter types and to generate the specific data types necessary to generate the stored procedure call. Additionally, the [TOKENS] section entries can supply an actual or default value to be used with development tools like query editors.

The TSA supports more than one method of parameter calling to allow flexibility with an order or precedence. The ExpandTokenizedSQL function scans the SQL parameters and changes any quote provided to the correct quote required by the connected DBMS. For example, if the DBMS requires double quotes around string values and the user supplied single quotes then the single quotes are changed to double quotes. This makes the supplied SQL statement more portable though some parameters for SQL server that identify tables, columns or keywords will want to stay in double quotes. Quotes embedded in a value that are the same as the DBMS delimiting quote will now be escaped by doubling the value. If table and column names are provided in the [TOKENS] section then use them as described If no tokens or quotes are provided then pass the parameter as is.

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 illustrates a development process diagram.

Figure 2 illustrates moving a tokenzed SQL to production mode.

Figure 3 illustrates tokenized SQL production deployment.

DETAILED DESCRIPTION OF THE INVENTION

1.1 SQT2SP utility usage

To convert SQT files to Stored Procedures run the SQT2SP utility by directing it to a folder of SQT files and a DSN. It will generate an output file that is ready to load all the appropriate stored procedures in the database server DSN specified. The developer then changes the global eSource variable in the appliction from the rddFile to the rddStoredProc value and the application will now run against the stored procedures. To support stored procedures, the SQT files need to have correctly specified entries. The [TOKENS] section should have all token names (without the leading @ sign) with each followed by an equal sign (=) and then a actual or sample value that is formatted for the correct data type (' or " for strings, nothing for numbers). The token entry will be preceded by a table.columnname value that matches the token names correct data types, if no table.columnname combination is available for the variable then the DBMS specific data type is declared (like VarChar(20) or int). If the CreateSP.exe has trouble converting certain SQT files it is because they do not meet the [TOKENS] section entry requirements.

For each file, it will first convert the SQT file to an XML doc that is a "flat" XML document, meaning it has sections and rows elements only, no real structured content. This file is then converted to a "SQTDef" XML doc in memory using an xslt style sheet 'SQTFile2SQTDef.xslt'. This work is done in 2 parts: phase 1 does everything but the database lookup to get data types for the parameters. Phase 2 hits the database for the data types and converts from vendor-specific DBMS data types to OLEDB data types. This is a more robust XML schema. This will then be validated with another xslt style sheet, valSQTDef.xslt. It's a touch harder to validate with xslt than with code (xslt doesn't do substrings very well), it's also more maintainable without recompiling. The "transformed" SQTDef xml doc is a plain text memory stream that is then written out to a log file if there are errors. This uses xslt extension objects that call back from xslt to C#. If the SQTDef is valid, there is a final conversion to a stored procedure with a vendor-specific xslt style sheet, of which there is an SQL Server example provided. A different XSLT style sheet is for each different database vendor. All stored procedures can get written to a single file or optionally written to one stored procedure file per SQT file. These outputted stored procedures file can the loaded into a query editor or command environment for the database and executed, generating all of the stored procedures required for that application.

1.2 Expand Tokenized SQL prototype

THOMAS UPDATE HERE:

```
RDDAPI(RC)          ExpandTokenizedSQL(
   IN    DBC *    pDBC,
   IN    long     fSource,
   IN    pszSourceSQL,
   OUT   char *   pszExpandedSQL,
   OUT   long *   pcbExpandedSQL,
    IN    char *   pszTokenValues
    );

    pDBC                    Pointer to a DBC.
                            This may be NULL for FILE and PARM sources.

    fSource                 Defines the source of the SQL statement. May
be:
                            SQL_SOURCE_FILE        Get the SQL from a file
                            SQL_SOURCE_TABLE       Get the SQL from a table
                            SQL_SOURCE_PARM        The SQL is passed in
pszSQLSource
                            SQL_SOURCE_PROC        The SQL is a stored
procedure.

    pszSourceSQL            String containing filename, record key or
SQL

    pszExpandedSQL          Pointer to a buffer for expanded SQL
statement.
                            This buffer is allocated by the caller.  You
may
                            call the function with this parameter set to
NULL and
                            the required buffer length will be returned
in
                            pcbExpandedSQL.

    pcbExpandedSQL          Length of the expanded SQL statement buffer.

    pszTokenValues          Pointer to Key=value; pairs for tokens.
                            Each key=value pair needs to be terminated
by a semi-colon.
```

Expand Tokenized SQL function usage

ExpandTokenizedSQL function calls returns an executable SQL statement string with any string tokens replaced by values specified in TokenValues parameter. Or it the stored procedure name then parameter list and this stored procedure parameter list will include the 'single quote' for the string parameter, date parameter etc since the stored procedures expects single quote with strings and date type input parameter list. For eSource = rddTable, the ActiveConnection property needs to be set with an active connection object. For eSource = rddFile, the SourceSQL can either be a fully qualified file name or file name with no path information. For a file name with no path information, the object assumes the file is located in the current vDir.

```
rc = rddExpandTokenizedSQL(pDBC,
                SQL_SOURCE_FILE | SQL_SOURCE_PROC,
"\\ADirectory\AFile.sqt",
&pszSQLBuffer,
&cbSQLBuffer,
"strOrgName=SIGN & strSignType=TYPE"
);
```

TokenValues parameter specifies the list of token/value pairs to replace tokenized parameters in the SourceSQL. This parameter string needs to be in the format:

token=value	for a single token/value pair
or	
token(1)=value(1)&token(2)=value(2)&...&token(n)=value(n)	for n token value pairs

The eSource parameter is an enumerated value which specifies the SQLSource type. This value is defaulted to rddString and is not required. Get the SQL source from the passed in parameter, a file, SGSQL table record for just build the stored procedure call. Possible eSource values are:

rddFile	the SQLSource is the name of a file or a fully qualified file name.
rddStoredProc	the SQLSource is the name of a stored procedure
rddString	the SQLSource is a SQL statement string
rddTable	the SQLSource is the name of a primary key value in SGSQL

Null string values are accepted by rddExpandTokenized SQL. When passing in the substitution values in the pszTokenValues parameter DO NOT pass in token value pairs that have no value and allow the default token to assign the value. Then when rddExpandTokenizedSQL is called pass in the tokens that have actual values, the others will be set to NULL by default if the NULL is specified in the [TOKENS] section for the token entry. This way SQL UPDATE or INSERT statements with tokens that may or may not have values can be used. Putting in two double quotes and double quotes around a space is difficult to read and may lose that token and all subsequent tokens. This way the ExpandTokenizedSQL function will provide the necessary values and something in the statement to indicate "this field empty" to the DB.

For example:

```
[TOKENS]
BUS_SCHEDULE.BUS_ROUTE BusRoute = NULL
```

Function Internals:

1. Check all parameters for valid values.
2. Read the tokenized SQL source from a file, table, or string buffer
3. If the source is in a file or table the [TOKENS] entry may be included in the source If they are present then read and concatenate with pszTokenValues.
4. Expanded the tokenized SQL, substituting token values where supplied

1.3 Sample database table named BUS_SCHEDULE

BUS_ROUTE	ROUTE_DIRECTION	BUS_STOP	ARRIVAL_TIME
NUMERIC(4)	VARCHAR(20)	VARCHAR(40)	DATETIME
8	Southbound	15th Ave and 80 St	8:07 AM
8	Southbound	15th Ave and 65 St	8:10 AM
8	Southbound	15th Ave and 45 St	8:13 AM
8	Southbound	15th Ave and Main St	8:20 AM

1.4 Example SQL program without TSA

```
'Initialize the MS ADO connection object and open a new connection
Dim oConn as New ADODB.Connection
oConn.Open "driver={SQL Server};server=DB_SERVER;user
id=sa;password=;database=BUS_SCHEDULE;"

'Initialize the recordset object and open a new recordset using the connection and SQL
statement
Dim rsBus as New ADODB.RecordSet

'Build the SQL Statement using variables for the bus route
'normally these  variables are determined by some user interface calls
Dim numBusRoute = 8
Dim strRouteDirection = "Southbound"

'note the included quotes around the RoutDirection variable
Dim strSQLstatement = "SELECT * FROM BUS_SCHEDULE WHERE BUS_ROUTE = " &
numBusRoute & " ROUTE_DIRECTION = '" & strRouteDirection & "'"
Set rsBus = oConn.Execute(strSQLstatement)

'THE REST IS THE SAME FOR ALL EXAMPLES
'walk the recordset, printing the route, stop and arrival time
While Not rsBus.EOF
  Debug.Print "Route: #" & rsBus("BUS_ROUTE") & " " & rsBus("ROUTE_DIRECTION") & "
Stop: " & rsBus("BUS_STOP") & "  Time: "& rsBus("ARRIVAL_TIME")
    rsBus.MoveNext
Wend
rsBus.Close
oConn.Close
```

1.5 Sample Tokenized SQL (SQT) file used with example

```
BUS_ROUTE.SQT

[SQL]
SELECT * FROM BUS_SCHEDULE
WHERE BUS_ROUTE = @BusRoute AND ROUTE_DIRECTION = @RouteDirection

[TOKENS]
BUS_SCHEDULE.BUS_ROUTE          BusRoute= 0
BUS_SCHEDULE.ROUTE_DIRECTION  RouteDirection = 'Northbound'

[COMMENTS]
select statement for use with printing a bus schedule
```

1.6 Example SQL program with SQT file

```
'Initialize the AccessVia database connection object and open a new connection
Dim oConn as New RDD.Connection
oConn.Open "driver={SQL Server};server=DB_SERVER;user
id=sa;password=;database=BUS_SCHEDULE;"


'Initialize the command object to parse the SQT file and open a new recordset using the
connection and SQL statement
Dim oCommand as New RDD.Command

'Build the SQL Statement from the SQT file using variables for the bus route
'normally these  variables are determined by some user interface calls
Dim numBusRoute = 8
Dim strRouteDirection = "Southbound"

'normally this eSource variable is defined globally from a configuration table or file
Dim eSource = rddFile   'source for SQT statement is from file, not string or stored procedure

'note that no quotes are required around the route direction value
Dim vTokens = "BusRoute=" & numBusRoute & "&RouteDirection=" & strRouteDirection
Dim strSQLstatement =
oCommand.ExpandTokenizedSQL("BUS_ROUTE.SQT",vTokens,eSource)
oCommand.Close


'Initialize the recordset object and open a new recordset using the connection and SQL
statement
Dim rsBus as New RDD.RecordSet
Set rsBus = oConn.Execute(strSQLstatement)


'THE REST IS THE SAME AS EXAMPLE 5.5
'walk the recordset, printing the route, stop and arrival time
```

```
While Not rsBus.EOF
  Debug.Print "Route: #" & rsBus("BUS_ROUTE") & " " & rsBus("ROUTE_DIRECTION") & " Stop: " & rsBus("BUS_STOP") & "  Time: "& rsBus("ARRIVAL_TIME")
    rsBus.MoveNext
Wend
rsBus.Close
oConn.Close
```

1.7 Example Stored Procedure program without TSA

```
'Initialize the MS ADO connection object and open a new connection
Dim oConn as New ADODB.Connection
oConn.Open "driver={SQL Server};server=DB_SERVER;user id=sa;password=;database=BUS_SCHEDULE;"

' Open a command object for a stored procedure with two parameters, note this code is unique
Dim cmdBus as New ADODB.Command
Set cmdBus.ActiveConnection = oConn
cmdBus.CommandText = "BUS_ROUTE"
cmdBus.CommandType = adCmdStoredProc
cmdBus.Parameters.Refresh

' Get parameter value, execute the command and store the results in a recordset
'normally these  variables are determined by some user interface calls
cmdBus.Parameters(1) = 8
cmdBus.Parameters(2) = "Southbound"

'Initialize the recordset object and open a new recordset using the command object
Set rsBus = cmdBus.Execute()

'THE REST IS THE SAME AS EXAMPLE 5.5
 'walk the recordset, printing the route, stop and arrival time
While Not rsBus.EOF
  Debug.Print "Route: #" & rsBus("BUS_ROUTE") & " " & rsBus("ROUTE_DIRECTION") & " Stop: " & rsBus("BUS_STOP") & "  Time: "& rsBus("ARRIVAL_TIME")
    rsBus.MoveNext
Wend
rsBus.Close
oConn.Close
```

1.8 Sample Stored Procedure used with example

```
CREATE PROCEDURE BUS_ROUTE (  @BusRoute as Numeric(2), @RouteDirection as VarChar(20) )
```

```
AS
BEGIN

SELECT * FROM BUS_SCHEDULE
WHERE BUS_ROUTE = @BusRoute AND ROUTE_DIRECTION = @RouteDirection

END

GO
```

1.9 Example Stored Procedure program with TSA

```
'Initialize the AccessVia database connection object and open a new connection
Dim oConn as New RDD.Connection
oConn.Open "driver={SQL Server};server=DB_SERVER;user
id=sa;password=;database=BUS_SCHEDULE;"

'Initialize the command object to parse the SQT file and open a new recordset using the
connection and SQL statement
Dim oCommand as New RDD.Command

'Build the SQL Statement from the SQT file using variables for the bus route
'normally these  variables are determined by some user interface calls
Dim numBusRoute = 8
Dim strRouteDirection = "Southbound"

'normally this eSource variable is defined globally from a configuration table or file
 Dim eSource = rddStoredProc  'source for SQT statement is from stored procedure
'THE ABOVE LINE IS THE ONLY DIFFERENCE FROM EXAMPLE 5.7

Dim vTokens = "BusRoute=" & numBusRoute & "&RouteDirection=" & strRouteDirection
Dim strSQLstatement =
oCommand.ExpandTokenizedSQL("BUS_ROUTE.SQT",vTokens,eSource)
oCommand.Close

 'Initialize the recordset object and open a new recordset using the connection and SQL
statement
Dim oRS as New RDD.RecordSet
Set rsBus = oConn.Execute(strSQLstatement)

'THE REST IS THE SAME AS EXAMPLE 5.5
'walk the recordset, printing the route, stop and arrival time
While Not rsBus.EOF
  Debug.Print "Route: #" & rsBus("BUS_ROUTE") & " " & rsBus("ROUTE_DIRECTION")  &  "
Stop: " & rsBus("BUS_STOP") & "  Time: "& rsBus("ARRIVAL_TIME")
```

```
    rsBus.MoveNext
Wend
rsBus.Close
oConn.Close
```

1.10 Source Code and XSLT for SQT2SP utility

The following source code is used to take an SQT file and a database connection, and generate database specific stored procedures

1.10.1 MS C# Source Code for SQT2SP executable

```
using System;
using System.Drawing;
using System.Collections;
using System.Collections.Specialized;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
using System.Xml;
using System.Xml.Xsl;
using System.IO;
using System.Data.OleDb;
using System.Diagnostics;
using System.Text.RegularExpressions;
/* non-MS namespaces */
using IniFile;

namespace SQT2StoredProc
{
  /// <summary>
  /// Summary description.
  /// </summary>
  public class SQT2XMLTransform
  {
    /* fields */

    private string m_SQTSourceDir;
    private string m_StoredProcOutputDir;
    private string m_StoredProcOutputFile = "CreateSQTStoredProcs.sql";
    //private string m_StoredProcOutputFile = "*";
    private string m_XsltDir;
    private string m_SQTFile2SQTDefXslt = "SQTFile2SQTDef.xslt";
    private string m_SQTDef2StoredProcXslt;
    protected XslTransform m_SQTFile2SQTDefTransform;
    protected XslTransform m_SQTDef2StoredProcTransform;
    protected XslTransform m_SQTDefValidator;
    private OleDbConnection m_dbConnection;
    protected string m_connectionString =
      "Provider=SQLOLEDB;Data Source=localhost;Initial
Catalog=WDSS;Integrated Security=SSPI;";

    /* properties */

    public string SQTSourceDir
    {
      get {return m_SQTSourceDir;}
      set {m_SQTSourceDir = value;}
    }
    public string StoredProcOutputDir
```

```
      {
        get {return m_StoredProcOutputDir;}
        set {m_StoredProcOutputDir = value;}
      }
      public string StoredProcOutputFile
      {
        get {return m_StoredProcOutputFile;}
        set {m_StoredProcOutputFile = value;}
      }
      public string XsltDir
      {
        get {return m_XsltDir;}
        set {m_XsltDir = value;}
      }
      public string SQTFile2SQTDefXslt
      {
        get {return m_SQTFile2SQTDefXslt;}
        set {m_SQTFile2SQTDefXslt = value;}
      }
      public string SQTDef2StoredProcXslt
      {
        get {return m_SQTDef2StoredProcXslt;}
        set {m_SQTDef2StoredProcXslt = value;}
      }
      public string ConnectionString
      {
        get {return m_connectionString;}
        set {m_connectionString = value;}
      }


      /* Constructors */

      public SQT2XMLTransform(string inSQTSourceDir, string
  inStoredProcOutputDir, string inXsltDir, string inSQTDef2StoredProcXslt,
  string inSQTDefValidator, string inConnectionString)
      {
        try
        {
          m_SQTSourceDir = inSQTSourceDir;
          StoredProcOutputDir = inStoredProcOutputDir;
          XsltDir = inXsltDir;
          ConnectionString = inConnectionString;
          m_SQTDef2StoredProcXslt = inSQTDef2StoredProcXslt;
          m_SQTFile2SQTDefTransform = new XslTransform();
          m_SQTFile2SQTDefTransform.Load(XsltDir + SQTFile2SQTDefXslt);
          m_SQTDefValidator = new XslTransform();
          m_SQTDefValidator.Load(XsltDir + inSQTDefValidator);
          m_SQTDef2StoredProcTransform = new XslTransform();
          m_SQTDef2StoredProcTransform.Load(XsltDir + SQTDef2StoredProcXslt);
          m_dbConnection = new OleDbConnection();
        }
        catch(Exception e)
        {
          Trace.WriteLine(System.DateTime.Now.ToString("F") + ":
  SQT2StoredProc: " + e.Message + e.StackTrace);
          Application.Exit();
        }
      }

      public static void Main(string[] args)
      {

        /* Set up error logging */
        Trace.Listeners.Add(new
          TextWriterTraceListener(File.Create(Application.StartupPath + @"\" +
  "SQT2StoredProc.log")));
        Trace.AutoFlush = true;
      Trace.Write("");
      Trace.Indent();

      Trace.WriteLine(System.DateTime.Now.ToString("F") + ": SQT2StoredProc:
Program Start");
      /* Read INI file from command line. Default is SQT2StoredProc.INI in
same folder as exe file */
      string strINIFileDir;
      if (args.Length == 0)
        strINIFileDir = Application.StartupPath + @"\" +
"SQT2StoredProc.INI";
      else
        strINIFileDir = args[0];
      IniFileReader ifr = new IniFileReader(strINIFileDir, true);
      /* Read INI file entries and instantiate a transformer class */
      string strSQTSourceDir = ifr.GetIniValue("Startup",
"SQTSourceFolder");
      if (!strSQTSourceDir.EndsWith(@"\"))
        strSQTSourceDir = strSQTSourceDir + @"\";
      string strStoredProcOutputDir = ifr.GetIniValue("Startup",
"StoredProcOutputFolder");
      string strXsltFolder = ifr.GetIniValue("Startup", "XsltFolder");
      string strSQTDef2StoredProcXsltFileName = ifr.GetIniValue("Startup",
"SQTDef2StoredProcXsltFileName");
      string strDBConnectionString = ifr.GetIniValue("Startup",
"DBConnectionString");
      string strSQTDefValidator = ifr.GetIniValue("Startup",
"SQTDefValidatorXsltFileName");
      SQT2XMLTransform transformer =
        new SQT2XMLTransform(strSQTSourceDir,
                   strStoredProcOutputDir,
                   strXsltFolder,
                   strSQTDef2StoredProcXsltFileName,
                   strSQTDefValidator,
                   strDBConnectionString);
      transformer.Transform();
      Trace.Unindent();
      Trace.WriteLine(System.DateTime.Now.ToString("F") + ": SQT2StoredProc:
Program End");
    }
    public bool Transform()
    {
      /*
       * Process the source directory in a loop, reading each SQT file and
transforming
       * first, instantiate objects and do some setup work outside the loop,
though.
      */
      try
      {
        DirectoryInfo SQTDirInfo = new DirectoryInfo(SQTSourceDir);
        //FileInfo[] SQTFileList = SQTDirInfo.GetFiles("*.SQT");
        //FileInfo[] SQTFileList = SQTDirInfo.GetFiles("Delete*.SQT");
        //FileInfo[] SQTFileList =
SQTDirInfo.GetFiles("DeleteSelectedSignStoreSignIDs.SQT");
        FileInfo[] SQTFileList = SQTDirInfo.GetFiles("AddMemberL1.SQT");
        //FileInfo[] SQTFileList =
SQTDirInfo.GetFiles("AddCopySignsForBatch.SQT");
        XmlDocument SQTDefXmlDoc = new XmlDocument();
        MemoryStream SQTDefMemStream = new MemoryStream();
        FileStream StoredProcSingleStream =
          new FileStream(StoredProcOutputDir + StoredProcOutputFile,
          FileMode.Create,
          FileAccess.Write,
          FileShare.Read);
        StreamWriter StoredProcOutWriter = new
StreamWriter(StoredProcSingleStream);
        OpenDbConnection(this.ConnectionString);
        foreach (FileInfo SQTFile in SQTFileList)
        {
          /*
           * First reads the INI-file-like SQT file from disk and convert it
to a
           * hierarchical, schema-based XML format in a memory stream.
          */
          Trace.WriteLine(System.DateTime.Now.ToString("F") + ":
SQT2StoredProc: Processing file " + SQTFile.Name);
          Trace.Indent();
          ConvertSQTFileToXML(SQTSourceDir + SQTFile.Name, ref
SQTDefMemStream);
          /*
           * Load the stream to an XmlDocument, then update the parameter
datatypes in
           * the SQTDef with information from the target database using
OleDB.
          */
          SQTDefXmlDoc.Load(SQTDefMemStream);
          UpdateParameterDataTypes(ref SQTDefXmlDoc);

//SQTDefXmlDoc.Save("E:\\data\\Source\\vbs\\SQT2SP\\sqtFiles\\SQTDef.XML");
          /*
           * Edit SQTDef xml document using XSLT instead of schemas, so we
can control
           * the processing and error messages, and write to a log file. If
there are errors,
           * skip to the next file.
          */
          if (!SQTDefIsValid(ref SQTDefXmlDoc))
          {
            Trace.Unindent();
            continue;
          }
          /*
           * The SQTDef is now complete. Transform it to actual stored
procedure code
           * for the target database. The XSLT stylesheet outputs to a non-
XML stream.
          */
          MemoryStream StoredProcMemStream = new MemoryStream();
          m_SQTDef2StoredProcTransform.Transform(SQTDefXmlDoc, null,
StoredProcMemStream);
          StoredProcMemStream.Position = 0;
          /*
           *
          */
//          if (StoredProcOutputFile == "*")
//          {
//            string[] outFileName = SQTFile.Name.Split('.');
//            outFileName[1] = ".SQL";
//            FileStream StoredProcOutStream =
//              new FileStream(StoredProcOutputDir + outFileName[0] + "." +
outFileName[1],
//              FileMode.Create,
//              FileAccess.Write,
//              FileShare.Read);
//          }
          /* Output the stream to persistent storage on disk */
          StreamReader StoredProcReader = new
StreamReader(StoredProcMemStream);
          while (StoredProcReader.Peek() > -1)
          {
            StoredProcOutWriter.WriteLine(StoredProcReader.ReadLine());
          }
          StoredProcReader.Close();
//          if (StoredProcOutputFile == "*")
//          {
//            StoredProcOutStream.Close();
//          }
          Trace.Unindent();
        }
        StoredProcOutWriter.Close();
        CloseDbConnection();
      } // end try block
      catch(Exception e)
      {
        Trace.WriteLine(System.DateTime.Now.ToString("F") + ":
SQT2StoredProc: " + e.Message + e.StackTrace);
      }

      return true;
    }
    protected void OpenDbConnection(string inConnectionString)
    {
      m_dbConnection.ConnectionString = ConnectionString;
      try
      {
        m_dbConnection.Open();
      }
      catch(Exception e)
      {
        Trace.WriteLine(System.DateTime.Now.ToString("F") + ":
SQT2StoredProc: " + e.Message + e.StackTrace);
      }
    }
    protected void CloseDbConnection()
    {
      m_dbConnection.Close();
    }
    protected void ConvertSQTFileToXML(string inFile, ref MemoryStream
inStream)
    {
      try
      {
        /*
        * Create an XML document from the SQT file. The schema for this
document is a general
        * INI-file-like schema with sections, and items that have 'key' and
'value attributes.
        */
        IniFileReader ifr = new IniFileReader(inFile, true);
        /* Uncomment the following  statements as needed for debugging.
         *
         * To Save to a File:
        */
         //ifr.OutputFilename = "<your output.XML>";
         //ifr.Save();
        /*
         * To Display:
        */
        //MessageBox.Show(ifr.XML);
        /*        /*
        * SQTFile2SQTDef stylesheet takes the base of the filename as a
parameter.
        * This will eventually become the name of the stored procedure.
        */
        XsltArgumentList xslArgs = new XsltArgumentList();
        FileInfo fileInfo = new FileInfo(inFile);
        string fileName = fileInfo.Name;
        string[] baseName = fileName.Split('.');
        xslArgs.AddParam("inSQTName", "", baseName[0]);
        /* Make sure to reset the stream properly for reuse in the loop:
shrink its
```

```
          * length to zero and put the stream pointer at the beginning BEFORE
processing,
          * and set the position back to zero AFTER transforming. This is
important!
         */
         inStream.SetLength(0);
         inStream.Seek(0, SeekOrigin.Begin);
         m_SQTFile2SQTDefTransform.Transform(ifr.XmlDoc, xslArgs, inStream);
         inStream.Position = 0; // do this so the stream can be read later
       }
       catch(Exception e)
       {
         Trace.WriteLine(System.DateTime.Now.ToString("F") + ":
SQT2StoredProc: e.Message + e.StackTrace");
       }
     }

     protected void UpdateParameterDataTypes(ref XmlDocument inXmlDoc)
     {
       OleDbType OleDbTypeInstance = new OleDbType();
       string strOleDbDataType = "";
       short shortNumericScale = 0;
       int intNumericPrecision = 0;
       long lDatetimePrecision = 0;
       long lCharMaxLength = 0;
       long lColumnFlags = 0;
       /* const values DBCOLUMNFLAGSENUM enum in OleDb.h; couldn't figger out
how to include in
        * C#, what class they belong to, etc. I hope they don't change!
       */
       const long lIsLong = 0x80;
       const long lIsFixedLength = 0x10;
       const long lScaleIsNegative = 0x4000;
       long fixedLengthTest;
       long longTest;
       long ScaleIsNegativeTest;

       XmlNodeList SQLColumnRefNodes =

inXmlDoc.DocumentElement.SelectNodes("SQTParameterList/SQTParameter/SQLColum
nRef");
       foreach(XmlNode SQLColumnRefNode in SQLColumnRefNodes)
       {
         string strTableName =
SQLColumnRefNode.SelectSingleNode("SQLTable").Attributes["name"].Value;
         string strColumnName =
SQLColumnRefNode.SelectSingleNode("SQLColumn").Attributes["name"].Value;
         Object[] columnFilter = {null, null, strTableName, strColumnName};
         /* Now get schema information from the database itself. */
         DataTable myColumnInfo =
m_dbConnection.GetOleDbSchemaTable(OleDbSchemaGuid.Columns, columnFilter);
         foreach(DataRow myRow in myColumnInfo.Rows)
         {
           strOleDbDataType =
Enum.GetName(OleDbTypeInstance.GetType(),myRow["DATA_TYPE"]);
           if (myRow["NUMERIC_SCALE"] != System.DBNull.Value)
             shortNumericScale = (short) myRow["NUMERIC_SCALE"];
           if (myRow["NUMERIC_PRECISION"] != System.DBNull.Value)
             intNumericPrecision = (int) myRow["NUMERIC_PRECISION"];
           if (myRow["DATETIME_PRECISION"] != System.DBNull.Value)
              lDatetimePrecision = (long) myRow["DATETIME_PRECISION"];
           if (myRow["CHARACTER_MAXIMUM_LENGTH"] != System.DBNull.Value)
             lCharMaxLength = (long) myRow["CHARACTER_MAXIMUM_LENGTH"];
           if (myRow["COLUMN_FLAGS"] != System.DBNull.Value)
             lColumnFlags = (long) myRow["COLUMN_FLAGS"];
         }
         fixedLengthTest = lColumnFlags & lIsFixedLength;
         longTest = lColumnFlags & lIsLong;
        ScaleIsNegativeTest = lColumnFlags & lScaleIsNegative;
        /* Update OleDbDataType attributes based on what was found in the
database */
        XmlNode dataTypeNode =
SQLColumnRefNode.SelectSingleNode("SQLColumn/OleDbDataType");
        dataTypeNode.Attributes["name"].Value = strOleDbDataType;
        if (shortNumericScale > 0)
          dataTypeNode.Attributes["numericScale"].Value =
shortNumericScale.ToString();
        if (intNumericPrecision > 0)
          dataTypeNode.Attributes["numericPrecision"].Value =
intNumericPrecision.ToString();
        if (lDatetimePrecision > 0)
          dataTypeNode.Attributes["dateTimePrecision"].Value =
lDatetimePrecision.ToString();
        if (lCharMaxLength > 0)
          dataTypeNode.Attributes["characterLength"].Value =
lCharMaxLength.ToString();
        if (fixedLengthTest == 0x10)
          dataTypeNode.Attributes["isFixedLength"].Value = "true";
        else
          dataTypeNode.Attributes["isFixedLength"].Value = "false";
        if (longTest == 0x80)
          dataTypeNode.Attributes["isLong"].Value = "true";
        else
          dataTypeNode.Attributes["isLong"].Value = "false";
        if (ScaleIsNegativeTest == 0x4000)
          dataTypeNode.Attributes["scaleIsNegative"].Value = "true";
        else
          dataTypeNode.Attributes["scaleIsNegative"].Value = "false";
        /* clear values */
        lCharMaxLength = 0;
        shortNumericScale = 0;
        intNumericPrecision = 0;
        lDatetimePrecision = 0;
        fixedLengthTest = 0;
        longTest = 0;
        ScaleIsNegativeTest = 0;
      }
      //MessageBox.Show(inXmlDoc.OuterXml);
    }

    protected bool SQTDefIsValid(ref XmlDocument inXmlDoc)
    {
      MemoryStream errorLogStream = new MemoryStream();
      XsltArgumentList xslArgs = new XsltArgumentList();
      xslArgs.AddExtensionObject("urn:SQT2StoredProc", this);
      m_SQTDefValidator.Transform(inXmlDoc, xslArgs, errorLogStream);
      if (errorLogStream.Length > 3)
      {
        /* Output the stream to the log file */
        StreamReader sr = new StreamReader(errorLogStream);
        errorLogStream.Position = 0;
        while (sr.Peek() > -1)
        {
          Trace.WriteLine(sr.ReadLine());
        }
        sr.Close();
        return false;
      }
      else
        return true;
    }

    public string paramNameIsValid(string inString)
    {

      Regex regex = new Regex("[a-zA-Z][a-zA-Z0-9]*");
      return regex.Match(inString).Value;
    }
  }
}
```

1.10.2 SQT2SP XSLT XML docouments

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSPY v5 rel. 2 U (http://www.xmlspy.com) by William
Barnum (Personal Copy) -->
<!-- ================================-->
<!-- Internal general entities for boilerplate     -->
<!-- ================================-->
<!DOCTYPE xsl:stylesheet [
<!ENTITY newLine "&#xd;&#xa;">
]>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="text" version="1.0" encoding="UTF-8" indent="no"/>
  <!-- ================================-->
  <!-- key lookups                                          -->
  <!-- ================================-->
  <!-- no keys in this stylesteet -->
  <!-- ================================-->
  <!--global variables                                      -->
  <!-- ================================-->
  <!-- no globals in this stylesteet -->
  <!-- ================================-->
  <!-- root template                                         -->
  <!-- ================================-->
  <xsl:template match="/">
    <xsl:text>CREATE PROCEDURE </xsl:text>
    <xsl:value-of select="/SQTDef/@name"/>
    <xsl:text> (</xsl:text>
    <xsl:call-template name="CreateParameterList"/>
    <xsl:text>)&newLine;</xsl:text>
    <xsl:text>AS&newLine;</xsl:text>
    <xsl:text>BEGIN&newLine;</xsl:text>
    <xsl:call-template name="CreateSQLStmtList"/>
    <xsl:text>END&newLine;</xsl:text>
    <xsl:text>GO&newLine;</xsl:text>
  </xsl:template>
  <!-- ================================-->
  <!--  CreateParameterList-->
  <!-- ================================-->
  <xsl:template name="CreateParameterList">
    <xsl:for-each select="//SQTParameter">
      <xsl:text>&#x40;</xsl:text>
      <xsl:value-of select="@name"/>
      <xsl:text> AS </xsl:text>
      <xsl:call-template name="OleDbDataType2SQLServer">
        <xsl:with-param name="inSQLColumn" select="SQLColumnRef/SQLColumn"/>
      </xsl:call-template>
      <xsl:if test="position() != last()">, </xsl:if>
    </xsl:for-each>
  </xsl:template>
  <!-- ================================-->
  <!-- OleDbDataType2SQLServer                    -->
  <!-- ================================-->
  <xsl:template name="OleDbDataType2SQLServer">
    <xsl:param name="inSQLColumn"/>
    <xsl:variable name="vName" select="$inSQLColumn/OleDbDataType/@name"/>
    <xsl:variable name="vNumericScale"
select="$inSQLColumn/OleDbDataType/@numericScale"/>
    <xsl:variable name="vNumericPrecision"
select="$inSQLColumn/OleDbDataType/@numericPrecision"/>
    <xsl:variable name="vDateTimePrecision"
select="$inSQLColumn/OleDbDataType/@dateTimePrecision"/>
```

```
    <xsl:variable name="vCharacterLength"
select="$inSQLColumn/OleDbDataType/@characterLength"/>
    <xsl:variable name="vIsFixedLength"
select="$inSQLColumn/OleDbDataType/@isFixedLength"/>
    <xsl:variable name="vIsLong"
select="$inSQLColumn/OleDbDataType/@isLong"/>
    <xsl:variable name="vScaleIsNegative"
select="$inSQLColumn/OleDbDataType/@scaleIsNegative"/>
    <xsl:choose>
      <xsl:when test="$vName='Char'">
        <xsl:choose>
          <xsl:when test="$vIsFixedLength='true'">
            <xsl:value-of select="concat('char(', $vCharacterLength, ')')"/>
          </xsl:when>
          <xsl:when test="$vIsLong='false'">
            <xsl:value-of select="concat('varchar(', $vCharacterLength,
')')"/>
          </xsl:when>
          <xsl:otherwise>text</xsl:otherwise>
        </xsl:choose>
      </xsl:when>
      <xsl:when test="$vName='WChar'">
        <xsl:choose>
          <xsl:when test="$vIsFixedLength='true'">
            <xsl:value-of select="concat('nchar(', $vCharacterLength,
')')"/>
          </xsl:when>
          <xsl:when test="$vIsLong='false'">
            <xsl:value-of select="concat('nvarchar(', $vCharacterLength,
')')"/>
          </xsl:when>
          <xsl:otherwise>ntext</xsl:otherwise>
        </xsl:choose>
      </xsl:when>
      <xsl:when test="$vName='Integer'">int</xsl:when>
      <xsl:when test="$vName='BigInt'">bigint</xsl:when>
      <xsl:when test="$vName='DBTimeStamp'">datetime</xsl:when>
      <xsl:when test="$vName='Boolean'">bit</xsl:when>
      <xsl:when test="$vName='Numeric'">
        <xsl:variable name="vTempScale">
          <xsl:choose>
            <xsl:when test="$vNumericScale='none'">0</xsl:when>
            <xsl:otherwise><xsl:value-of
select="$vNumericScale"/></xsl:otherwise>
          </xsl:choose>
        </xsl:variable>
        <xsl:value-of select="concat('numeric(', $vNumericPrecision, ',',
$vTempScale, ')')"/>
      </xsl:when>
      <xsl:when test="$vName='Binary'">
        <xsl:choose>
          <xsl:when test="$vIsFixedLength='true'">
            <xsl:choose>
              <xsl:when test="$vCharacterLength='8'">timestamp</xsl:when>
              <xsl:otherwise>
                <xsl:value-of select="concat('binary(', $vCharacterLength,
')')"/>
              </xsl:otherwise>
            </xsl:choose>
          </xsl:when>
          <xsl:otherwise>
            <xsl:choose>
              <xsl:when test="$vIsLong='true'">image</xsl:when>
              <xsl:otherwise>
                <xsl:value-of select="concat('varbinary(',
$vCharacterLength, ')')"/>
              </xsl:otherwise>
            </xsl:choose>
          </xsl:otherwise>
        </xsl:choose>
      </xsl:when>
      <xsl:when test="$vName='Currency'">
        <xsl:choose>
          <xsl:when test="$vNumericPrecision='19'">money</xsl:when>
          <xsl:otherwise>smallmoney</xsl:otherwise>
        </xsl:choose>
      </xsl:when>
      <xsl:when test="$vName='Double'">float</xsl:when>
      <xsl:when test="$vName='Single'">
        <xsl:choose>
          <xsl:when
test="$vNumericPrecision='none'">smalldatetime</xsl:when>
          <xsl:otherwise>real</xsl:otherwise>
        </xsl:choose>
      </xsl:when>
      <xsl:when test="$vName='SmallInt'">smallint</xsl:when>
      <xsl:when test="$vName='UnsignedTinyInt'">tinyint</xsl:when>
      <xsl:when test="$vName='Variant'">sql_variant</xsl:when>
      <xsl:when test="$vName='Guid'">uniqueid</xsl:when>
      <xsl:otherwise>
        <xsl:value-of select="concat('Unknown DataType: ', $vName)"/>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:template>
  <!-- ================================-->
  <!-- CreateSQLStmtList                              -->
  <!-- ================================-->
  <xsl:template name="CreateSQLStmtList">
    <xsl:for-each select="//SQLStmt">
      <xsl:value-of select="@text"/>
      <xsl:text>&newLine;</xsl:text>
    </xsl:for-each>
  </xsl:template>
  <!-- ================================-->
  <!--                                                                 -->
  <!-- ================================-->
</xsl:stylesheet>


<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSPY v5 rel. 2 U (http://www.xmlspy.com) by William
Barnum (Personal Copy) -->
<!-- ================================-->
<!-- Internal general entities for boilerplate     -->
<!-- ================================-->
<!DOCTYPE xsl:stylesheet [
<!ENTITY newLine "&#xd;&#xa;">
]>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>
  <xsl:param name="inSQTName" select="'Unknown'"/>
  <!-- ================================-->
  <!-- key lookups                                      -->
  <!-- ================================-->
  <xsl:key name="keyColumnRef"
match="sections/section[@name='COMMENTS']/item" use="@key"/>
  <!-- ================================-->
  <!--global variables                                  -->
  <!-- ================================-->
  <!-- no globals in this stylesteet -->
  <!-- ================================-->
  <!-- root template                                     -->
  <!-- ================================-->
  <xsl:template match="/">
    <SQTDef>
```

```
      <xsl:attribute name="name"><xsl:value-of
select="$inSQTName"/></xsl:attribute>
      <xsl:element name="SQTParameterList">
        <xsl:call-template name="SQTParameterList"></xsl:call-template>
      </xsl:element>
      <xsl:element name="SQLStmtList">
        <xsl:call-template name="SQLStmtList"></xsl:call-template>
      </xsl:element>
    </SQTDef>
  </xsl:template>
  <!-- ================================-->
  <!--  SQTParameterList                                  -->
  <!-- ================================-->
  <xsl:template name="SQTParameterList">
    <xsl:for-each select="sections/section[@name='TOKENS' or @name='DEFAULT
TOKENS']/item">
      <xsl:element name="SQTParameter">
        <xsl:call-template name="SQTParameter">
          <xsl:with-param name="inKey" select="@key"/>
        </xsl:call-template>
      </xsl:element>
    </xsl:for-each>
  </xsl:template>
  <!-- ================================-->
  <!-- SQTParameter                                         -->
  <!-- ================================-->
  <xsl:template name="SQTParameter">
    <xsl:param name="inKey"/>
    <xsl:variable name="varParamName">
      <xsl:choose>
        <xsl:when test="starts-with(normalize-space($inKey), 'NUMBER') or
starts-with(normalize-space($inKey), 'STRING')      ">
          <xsl:value-of select="substring-after(normalize-space($inKey), '
')"/>
        </xsl:when>
        <xsl:otherwise>
          <xsl:value-of select="normalize-space($inKey)"/>
        </xsl:otherwise>
      </xsl:choose>
    </xsl:variable>
    <xsl:variable name="varParamDataType">
      <xsl:choose>
        <xsl:when test="starts-with(normalize-space($inKey), 'NUMBER') or
starts-with(normalize-space($inKey), 'STRING')      ">
          <xsl:value-of select="substring-before(normalize-space($inKey), '
')"/>
        </xsl:when>
        <xsl:otherwise>
          <xsl:value-of select="'Unknown'"/>
        </xsl:otherwise>
      </xsl:choose>
    </xsl:variable>
    <xsl:attribute name="name"><xsl:value-of
select="$varParamName"/></xsl:attribute>
    <xsl:element name="SQTDataType"><xsl:attribute name="name"><xsl:value-of
select="$varParamDataType"/></xsl:attribute></xsl:element>
    <xsl:element name="SQLColumnRef">
      <xsl:call-template name="SQLColumnRef">
        <xsl:with-param name="inParamName" select="$varParamName"/>
      </xsl:call-template>
    </xsl:element>
  </xsl:template>
  <!-- ================================-->
  <!-- SQLColumnRef                                         -->
  <!-- ================================-->
  <xsl:template name="SQLColumnRef">
    <xsl:param name="inParamName"/>
    <xsl:variable name="varTableName" select="substring-
before(key('keyColumnRef', concat('@', $inParamName))/@value, '.')"/>
    <xsl:variable name="varColumnName" select="substring-
after(key('keyColumnRef', concat('@', $inParamName))/@value, '.')"/>
    <xsl:element name="SQLTable"><xsl:attribute name="name"><xsl:value-of
select="$varTableName"/></xsl:attribute></xsl:element>
    <xsl:element name="SQLColumn">
      <xsl:call-template name="SQLColumn">
        <xsl:with-param name="inColumnName" select="$varColumnName"/>
      </xsl:call-template>
    </xsl:element>
  </xsl:template>
  <!-- ===================================-->
  <!--  SQLColumn                                        -->
  <!-- ===================================-->
  <xsl:template name="SQLColumn">
    <xsl:param name="inColumnName"/>
      <xsl:attribute name="name"><xsl:value-of
select="$inColumnName"/></xsl:attribute>
      <xsl:element name="OleDbDataType">
        <xsl:attribute name="name"><xsl:value-of
select="'Unknown'"/></xsl:attribute>
        <xsl:attribute name="numericScale"><xsl:value-of
select="'none'"/></xsl:attribute>
        <xsl:attribute name="numericPrecision"><xsl:value-of
select="'none'"/></xsl:attribute>
        <xsl:attribute name="dateTimePrecision"><xsl:value-of
select="'none'"/></xsl:attribute>
        <xsl:attribute name="characterLength"><xsl:value-of
select="'none'"/></xsl:attribute>
        <xsl:attribute name="isFixedLength"><xsl:value-of
select="'Unknown'"/></xsl:attribute>
        <xsl:attribute name="isLong"><xsl:value-of
select="'Unknown'"/></xsl:attribute>
        <xsl:attribute name="scaleIsNegative"><xsl:value-of
select="'Unknown'"/></xsl:attribute>
      </xsl:element>
  </xsl:template>
  <!-- ===================================-->
  <!-- SQLStmtList                                        -->
  <!-- ===================================-->
  <xsl:template name="SQLStmtList">
    <xsl:for-each select="sections/section[@name='SQL']">
      <xsl:sort select="position()" data-type="number" order="ascending"/>
      <xsl:element name="SQLStmt">
        <xsl:call-template name="SQLStmt">
          <xsl:with-param name="inNode" select="."/>
        </xsl:call-template>
      </xsl:element>
    </xsl:for-each>
  </xsl:template>
  <!-- ===================================-->
  <!-- SQLStmt                                             -->
  <!-- ===================================-->
  <xsl:template name="SQLStmt">
    <xsl:param name="inNode"/>
    <xsl:variable name="varSQLText">
      <xsl:call-template name="concatSQLText">
        <xsl:with-param name="inElementList" select="item"/>
      </xsl:call-template>
    </xsl:variable>
    <xsl:attribute name="text"><xsl:value-of
select="$varSQLText"/></xsl:attribute>
  </xsl:template>
  <!-- ===================================-->
  <!-- concatSQLText                                   -->
  <!-- ===================================-->
  <xsl:template name="concatSQLText">
```

```
      <xsl:param name="inElementList"/>
      <xsl:choose>
        <xsl:when test="$inElementList">
          <!--
          The SQL will probably have equal signs in it, so concatenate key and
value attributes. Also
          preserve carriage returns & line feeds
          -->
          <xsl:variable name="varKey" select="normalize-
space($inElementList[position() = 1]/@key)"/>
          <xsl:variable name="varValue" select="normalize-
space($inElementList[position() = 1]/@value)"/>
          <xsl:variable name="varCurrentLine">
            <xsl:choose>
              <xsl:when test="$varValue">
                <xsl:value-of select="concat($varKey, '=', $varValue)"/>
                <!-- Don't add newLine chars on the last input line -->
                <xsl:if test="$inElementList[position() !=
last()]"><xsl:text>&newLine;</xsl:text></xsl:if>
              </xsl:when>
              <xsl:otherwise>
                <xsl:value-of select="$varKey"/>
                <!-- Don't add newLine chars on the last input line -->
                <xsl:if test="$inElementList[position() !=
last()]"><xsl:text>&newLine;</xsl:text></xsl:if>
              </xsl:otherwise>
            </xsl:choose>
          </xsl:variable>
          <xsl:variable name="varRemainingElements">
            <xsl:call-template name="concatSQLText">
              <xsl:with-param name="inElementList"
select="$inElementList[position() != 1]"/>
            </xsl:call-template>
          </xsl:variable>
          <xsl:value-of select="concat($varCurrentLine,
$varRemainingElements)"/>
        </xsl:when>
        <xsl:otherwise></xsl:otherwise>
      </xsl:choose>
    </xsl:template>
    <!-- ================================-->
    <!--                                                                  -->
    <!-- ================================-->
</xsl:stylesheet>


<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSPY v5 rel. 2 U (http://www.xmlspy.com) by William
Barnum (Personal Copy) -->
<!-- ***********************************************-->
<!-- valSQTDef.xslt: validate SQTDef doc     -->
<!--***********************************************-->



<!-- ================================-->
<!-- Internal general entities for boilerplate     -->
<!-- ================================-->
<!DOCTYPE xsl:stylesheet [
<!ENTITY newLine "&#xd;&#xa;">
]>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:SQT2StoredProc="urn:SQT2StoredProc">
    <xsl:output method="text" version="1.0" encoding="UTF-8" indent="no"/>
    <!-- ================================-->
    <!-- key lookups                                -->
    <!-- ================================-->
```

```
  <!-- =================================-->
  <!--global variables                                      -->
  <!-- =================================-->
  <!-- =================================-->
  <!-- root template                                         -->
  <!-- =================================-->
  <!-- =================================-->
  <!--  SQTDef                                               -->
  <!-- =================================-->
  <xsl:template match="SQTDef">
    <xsl:if test="string-length(@name)=0"><xsl:text>SQTDef element error:
missing name&newLine;</xsl:text></xsl:if>
    <xsl:apply-templates/>
  </xsl:template>
  <!-- =================================-->
  <!--  SQTParameterList                               -->
  <!-- =================================-->
  <xsl:template match="SQTParameterList">
    <xsl:apply-templates/>
  </xsl:template>
  <!-- =================================-->
  <!-- SQTParameter                                        -->
  <!-- =================================-->
  <xsl:template match="SQTParameter">
    <xsl:if test="string-length(@name)=0"><xsl:text>SQTParameter element
error: missing name&newLine;</xsl:text></xsl:if>
    -<xsl:if
test="SQT2StoredProc:paramNameIsValid(@name)=false()"><xsl:value-of
select="@name"/></xsl:if>
    <!--<xsl:value-of select="SQT2StoredProc:paramNameIsValid('@name')"/> --
>
    </xsl:template>
  <!-- =================================-->
  <!-- SQLColumnRef                                     -->
  <!-- =================================-->
  <xsl:template match="SQLColumnRef">
    <xsl:apply-templates/>
  </xsl:template>
  <!-- =================================-->
  <!-- SQLStmtList                                           -->
  <!-- =================================-->
  <xsl:template match="SQLStmtList">
    <xsl:apply-templates/>
  </xsl:template>
  <!-- =================================-->
  <!-- SQLStmt                                                 -->
  <!-- =================================-->
  <xsl:template match="SQLStmt">
    <xsl:apply-templates/>
  </xsl:template>
  <!-- =================================-->
  <!-- concatSQLText                                     -->
  <!-- =================================-->
  <xsl:template match="parseSQLText">
  </xsl:template>
  <!-- =================================-->
  <!--                                                              -->
  <!-- =================================-->
</xsl:stylesheet>
```

1.10.3 Sample INI file used with SQT2SP

```
[Startup]
SQTSourceFolder=E:\data\Source\vbs\SQT2SP\sqtFiles
StoredProcOutputFolder=E:\data\Source\vbs\SQT2SP\sqtFiles\
```

XsltFolder=E:\data\Visual Studio Projects\SQT2StoredProc\

SQTDef2StoredProcXsltFileName=SQTDef2SQLServerStoredProc.xslt

SQTDefValidatorXsltFileName=valSQTDef.xslt

DBConnectionString=Provider=SQLOLEDB;Data Source=localhost;Initial Catalog=WDSS;Integrated Security=SSPI;

;DBConnectionString=Provider=MSDAORA.1;User ID=myUID;password=myPWD;Data Source=myOracleServer;Persist Security Info=False";

1.11 'C' Source Code for ExpandTokenizedSQL

A note on the fonts used in this section

Courier New 9pt	Original source code
Times New Roman 12pt	Comments added to explain the source.

1.11.1 'C' Source Code for ExpandTokenizedSQL COM method

THOMAS UPDATE HERE:

1.11.2 'C' Source Code for rddExpandTokenizedSQL function

The code is laid out with the subroutines defined before the main API routine. This was done to avoid the need to forward reference the subroutine prototypes.

```
/*-----------------------------------------------------------------
---------
   Tokenized SQL routines
-----------------------------------------------------------------
-------*/
#define MAX_TOKEN            32
#define MAX_TOKEN_LEN        16
#define TokenDelimiter       '&'
#define BeginToken           '@'

#define TokenString          1
#define TokenNumber          2
#define TokenDate            3
#define TokenUnknown         4

typedef  struct   _Token
         {
         char *   pszToken;
         int      nDatatype;
         char *   pszValue;
         }
         Token,  *pToken;
```

_dbBuildTokenArrays parses the token/value pairs and builds a table of Token structures that will be used in the substitution pass of the SQL statement. The list of token/value pairs are a concatenation of the pszTokenValues parameter passed from the caller followed by the default token/values pairs found in the SQL source.

```
PRIVATE  RC        _dbBuildTokenArrays(
   IN    DBC *     pDBC,
   IN    char *    pszTokens,
   OUT   pToken    prgToken
   )
{
auto     int       iToken;
auto     int       iDefault;
auto     int       iScan;
auto     char *    pszTok;
auto     char *    psz;
static   char *    pszDatatype[] = {"STRING", "NUMBER", "DATE"};
auto     int       iDatatype;

auto     char      chQuoteDB = chOption(OPT_QUOTE_CHAR);
auto     char      chQuoteEX = chQuoteEX = chQuoteDB == '"' ?
(char)'\'' : (char)'"';
auto     char      rgchQuote[3];

   rgchQuote[0] = chQuoteDB;
   rgchQuote[1] = chQuoteEX;
   rgchQuote[2] = '\0';

/* Build an array of char * to the Key/Value pairs
*/
   MEMSET(prgToken, 0, sizeof(Token) * MAX_TOKEN);

   if (pszTokens[0] == '\0')
      return RDD_SUCCESS;

   pszTok = pszTokens;
   iToken = 0;
   do
      {
/*    See if the Token has a data type
*/
      for (iDatatype = 0; iDatatype < 3; ++iDatatype)
         {
         if (dstrstri(pszTok, pszDatatype[iDatatype]) == pszTok)
            {
            while (*pszTok != ' ') pszTok++;
            while (isspace(*pszTok)) ++pszTok;
            break;
            }
         }

      prgToken[iToken].nDatatype = iDatatype + 1;
/*    Get the KEY name
*/
      prgToken[iToken].pszToken = pszTok;
      while ((pszTok = strchr(pszTok, TokenDelimiter)) != NULL)
         {
/*       ----------------------------------------------------
         IF escaped delimiter THEN shift line left 1.
         Keep delimiter in the substitute value
         ----------------------------------------------------
*/
         if (pszTok[1] == TokenDelimiter)
            {
            strcpy(pszTok, pszTok+1);
            pszTok++;
            }
         else
            {
            *pszTok++ = '\0';
            break;
            }
         }

/*    Get the tokens substitution value
*/
      if ((psz = strchr(prgToken[iToken].pszToken, '=')) == NULL)
         return RDD_BAD_ARGUMENT_6;

      *psz++ = NULL;
      alltrim(prgToken[iToken].pszToken);

      alltrim(psz);
      prgToken[iToken].pszValue = psz;

      rtrim(prgToken[iToken].pszValue);
      }
   while (++iToken < MAX_TOKEN && pszTok);
```

```
/* ------------------------------------------------------------------
--------
   Try to fix up those tokens passed or default which don't have a
data type.
   This takes two passes through the token array.  The first pass
will set
   any defaults not explicitly set in the SQT coding.
   ------------------------------------------------------------------
------*/
   for (iScan = 0; iScan < 2; ++iScan)
      {
      for (iToken = 0; prgToken[iToken].pszToken != NULL; ++iToken)
         {
         if (prgToken[iToken].nDatatype == TokenUnknown)
            {
            for (iDefault = iToken+1; prgToken[iDefault].pszToken;
++iDefault)
               {
               if (dstricmp(prgToken[iToken].pszToken,
                            prgToken[iDefault].pszToken) == 0)
                  {
                  prgToken[iToken].nDatatype =
prgToken[iDefault].nDatatype;
                  break;
                  }
               }

            if (prgToken[iDefault].pszToken == NULL)
               {
               if (prgToken[iToken].pszValue[0] == chQuoteDB ||
                   prgToken[iToken].pszValue[0] == chQuoteEX
                  )
                  prgToken[iToken].nDatatype = TokenString;
               else
                  prgToken[iToken].nDatatype = TokenNumber;
               }
            }

   /*    If there are quotes around a non-string value THEN remove
*/
         if (prgToken[iToken].nDatatype != TokenString      &&
               (prgToken[iToken].pszValue[0] == chQuoteDB   ||
                prgToken[iToken].pszValue[0] == chQuoteEX
               )
            ){
            *strchr(prgToken[iToken].pszValue + 1,
                    prgToken[iToken].pszValue[0]) = ' ';
            prgToken[iToken].pszValue[0] = ' ';
            alltrim(prgToken[iToken].pszValue);
            }
         }
      }

   return RDD_SUCCESS;
}
```

_dbExpandTokens
This is the work horse routine that takes the SQL source and token value routine finds and parses the [TOKENS] section of a tokenized SQL statement. A list of token/value pairs is built and passed back as the return value. If parameters were also passed into the API those token/value pairs are placed at the beginning of the concatenated list.

```
PRIVATE  RC        _dbExpandTokens(
   IN    DBC *     pDBC,
   IN    char *    pszSQLSource,
   OUT   char *    pszExpandedSQL,
   OUT   long *    pcbExpandedSQL,
   IN    char *    pszTokenValues
   )
{
static   char      szDelimiters[] = " ,=\"\'\t\n\r()";
auto     char *    pszTokens;
auto     int       iToken;
auto     Token     rgToken[MAX_TOKEN];
auto     char      szToken[32];
auto     int       cbToken;
auto     char *    pszSave;
auto     char *    psz;
auto     char *    pszIN;
auto     char *    pszOUT;
auto     BOOL      bFoundToken;
auto     BOOL      bWork = FALSE;
auto     RC        rc    = RDD_SUCCESS;

auto     char      chQuoteDB = chOption(OPT_QUOTE_CHAR);
auto     char      chQuoteEX = chQuoteEX = chQuoteDB == '"' ?
(char)'\'' : (char)'"';


   if ((pszTokens = (char *)ALLOC(STRLEN(pszTokenValues) + 16)) ==
NULL)
      return RDD_NO_MEMORY;
   STRCPY(pszTokens, pszTokenValues);

   rc = _dbBuildTokenArrays(pDBC, pszTokens, rgToken);
   if (rc != RDD_SUCCESS)
      goto _Exit_Here;

   if (pszExpandedSQL == NULL)
      {
      bWork = TRUE;
      pszExpandedSQL = (char
*)ALLOC(KiloBytes(iOption(OPT_SQL_WORK_BUFFER)));
      }
   pszOUT = pszExpandedSQL;
   pszIN  = pszSQLSource;

/* Check for correct quote characters and change to quotes used by
DBMS    */
   _dbCheckQuotes(pDBC, pszSQLSource);
```

```
/* -------------------------------------------------------------------
--------
   All the work data is set up and the Token substitution array
built.

   Now go through the source SQL and copy to the Expanded buffer
inserting
   all token key values into the output stream.
   -----------------------------------------------------------------
------*/
   while (*pszIN)
      {
      if (*pszIN != BeginToken)
         {
         *pszOUT++ = *pszIN++;
         continue;
         }

/*    "Houston, we have a token" maybe
*/
      pszSave = pszIN++;                /* IF not valid we restore
*/
      cbToken = 0;

      for (psz = szToken; !strchr(szDelimiters, *pszIN); *psz++ =
*pszIN++)
         {
         ++cbToken;
         if (*pszIN == '\0' || cbToken == MAX_TOKEN_LEN)
            break;
         }
      *psz = '\0';                      /* Terminate the copied token
*/

      if (cbToken == 0)                 /* If an BeginToken alone
*/
         {
         *pszOUT++ = *pszSave++;        /* Copy the BeginToken
*/
         continue;
         }

/*    Find the token and substitute the value
*/
      bFoundToken = FALSE;
      for (iToken = 0; iToken < MAX_TOKEN; ++iToken)
         {
         if (rgToken[iToken].pszToken == NULL)
            break;

         if (dstricmp(rgToken[iToken].pszToken, szToken) == 0)
            {
            psz = rgToken[iToken].pszValue;

/*          --------------------------------------------------------
--------
            The value may have quote delimiters

            Check to make sure it is the correct quote char for the
            connected DBMS.  Change if needed else just copy to SQL
            --------------------------------------------------------
------*/
            if (*psz == chQuoteEX)        /* IF the first char is a
Quote */
               {
               *pszOUT++ = chQuoteDB;     /* Change to the correct
Quote   */
               psz++;
```

```
                        }
                     else
                     if (*psz == chQuoteDB)           /* IF the correct quote
        char      */
                        *pszOUT++ = *psz++;           /*    THEN just copy
        */
                     else
                     if (rgToken[iToken].nDatatype == TokenString)
                        *pszOUT++ = chQuoteDB;
```

```
/*            ---------------------------------------------------------
--------
              Copy the value characters to the SQL
              Escape any embedded quotes by doubling the quote
character
              ---------------------------------------------------------
------*/
              while (*psz)
                 {
                 if (psz[1] == '\0')          /* IF on the last character
*/
                    break;

                 if (*psz == chQuoteDB)       /* IF and embedded quote
*/
                    *pszOUT++ = chQuoteDB;    /*    THEN escape by
doubling      */

                 *pszOUT++ = *psz++;
                 }

/*            ---------------------------------------------------------
--------
              The value may have quote delimiters

              Check the last character like the first (see above)
              ---------------------------------------------------------
------*/
              if (*psz == chQuoteEX)          /* IF the first char is a
Quote  */
                 {
                 *pszOUT++ = chQuoteDB;
                 psz++;
                 }
              else
              if (*psz == '\0')               /* IF the data is null
*/
                 {
                 if (pszOUT[-1] == chQuoteDB)
                    {
                    --pszOUT;                 /* Remove the opening quote
*/
                    ++pszIN;                  /* Remove closing quote
*/
                    }

                 strcpy(pszOUT, "NULL");
                 pszOUT += 4;

                 break;
                 }
              else
                 *pszOUT++ = *psz++;          /*    THEN just copy
*/

              if (rgToken[iToken].nDatatype == TokenString &&
                    psz[-1] != chQuoteDB)
                 *pszOUT++ = chQuoteDB;

              break;
              }
           }

/*     IF not a valid token with substitution then copy as is
*/
       if (iToken == MAX_TOKEN || rgToken[iToken].pszToken == NULL)
          {
            --pszIN;
            while (pszSave < pszIN)
               *pszOUT++ = *pszSave++;
            }
         }

      *pcbExpandedSQL = pszOUT - pszExpandedSQL;
      *pszOUT++ = '\0';

   /* ----------- */
      _Exit_Here:
   /* ----------- */

      if (bWork)
         FREE(pszExpandedSQL);

      FREE(pszTokens);

      return rc;
   }
```

_dbGetDefaultTokens
This routine finds and parses the [TOKENS] section of a tokenized SQL statement. A list of token/value pairs is built and passed back as the return value. If parameters were also passed into the API those token/value pairs are placed at the beginning of the concatenated list.

```
PRIVATE  char *   _dbGetDefaultTokens(
   IN    char *   pszSourceSQL,
   IN    char *   pszTokenValues
   )
{
auto     char *   psz;
auto     char *   pszKeyWord;
auto     char *   pszDefaults;
auto     char *   pszAllTokens = NULL;
auto     char     szEndTokens[2] = {TokenDelimiter, 0};

   if ((pszKeyWord = dstrstri(pszSourceSQL, SECTION_TOKENS)) == NULL)
      return NULL;

   for (psz = pszKeyWord; *psz != '['; --psz)
      *psz = '\0';
   *psz = '\0';

   while (*psz != ']') *psz++ = '\0';
   *psz++ = '\0';

   while (isspace(*psz)) ++psz;

   pszDefaults = psz;

   while (*psz)
      {
      if (*psz == '\n')
         *psz = TokenDelimiter;
      else
      if (*psz == '\r')
         *psz = ' ';

      ++psz;
      }

   psz = &pszDefaults[strlen(pszDefaults) - 1];
   while (*psz == TokenDelimiter || *psz == ' ')
      *psz-- = '\0';

   pszAllTokens = (char *)ALLOC(strlen(pszTokenValues) +
strlen(pszDefaults) + 32);

   if (strlen(pszTokenValues))
      {
      strcpy(pszAllTokens, pszTokenValues);
      strcat(pszAllTokens, szEndTokens);
      }
   strcat(pszAllTokens, pszDefaults);

   return pszAllTokens;
}
```

_dbReadFileSQL

This routine reads a SQL statement from an external file. The filename can be a pathname or partial pathname.

```
PRIVATE  RC       _dbReadFileSQL(char ** ppszSourceSQL)
{
auto     FILE *   pfSQL;
auto     int      cb;

   if ((pfSQL = fopen(*ppszSourceSQL, "rb")) == NULL)
      return RDD_FILE_NOT_FOUND;

/* Get the size of the file
*/
   fseek(pfSQL, 0, SEEK_END);
   cb = ftell(pfSQL);
   fseek(pfSQL, 0, SEEK_SET);

   if ((*ppszSourceSQL = (char *)ALLOC(cb + 64)) == NULL)
      {
      fclose(pfSQL);
      return RDD_NO_MEMORY;
      }

/* Read the file into the allocated buffer
*/
   fread(*ppszSourceSQL, 1, cb, pfSQL);
   fclose(pfSQL);

   return RDD_SUCCESS;
}
```

_dbReadTableSQL

This routine reads a SQL statement from a table named SGSQL. The two columns that needs to be defined in the table are:

SQLName	contains the SQL name
SQLText.	contains the SQL statement.

```
PRIVATE  RC        _dbReadTableSQL(DBC * pDBC, char ** ppszSourceSQL)
{
auto     char      szQuery[256];
auto     char      szTable[64];
auto     char      szName[64];
auto     char *    psz;
auto     HTABLE    hTbl;
auto     RC        rc;
auto     OptFlag   fOpt = {TRUE};
auto     DBVALUE   dbValue;

   ValidateDBC(pDBC);

   psz = *ppszSourceSQL;
   if ((psz = strchr(psz, '.')) != NULL)
      {
      *psz++ = '\0';
      strcpy(szTable, *ppszSourceSQL);
      strcpy(szName,  psz);
      }
   else
      {
      strcpy(szTable, "SGSQL");
      strcpy(szName,  *ppszSourceSQL);
      }

   sprintf(szQuery,
      "SELECT SQLName, SQLText FROM %s WHERE SQLName = '%s'",
      szTable, szName
      );

   rc = rddOpenQuery(pDBC, "GetSQLSource", szQuery, fOpt, &hTbl);
   if (rc != RDD_SUCCESS)
      return rc;

   if ((rc = rddFetchRow(pDBC, hTbl, NULL, dbFirst)) == RDD_SUCCESS)
      {
      rc = rddGetData(pDBC, hTbl, "SQLText", NULL, &dbValue);

      psz = (char *)ALLOC(dbValue.sColumnWidth + 32);

      if (psz == NULL)
         rc = RDD_NO_MEMORY;
      else
         strcpy(psz, dbValue.pszChar);

      *ppszSourceSQL = psz;
      }

   rddCloseHandle(pDBC, hTbl);

   return rc;
}
```

_dbFormatProcedureCall
If the call requested that a stored procedure be executed then this routine is called. The routine builds the EXECUTE statement adding the procedure name and its parameters and passes this EXECUTE statement in the expanded SQL buffer.
```
PRIVATE  RC       _dbFormatProcedureCall(
   IN    DBC *    pDBC,
   IN    char *   pszSQLSource,
   OUT   char *   pszExpandedSQL,
   OUT   long *   pcbExpandedSQL,
   IN    char *   pszTokenValues
   )
{
auto     Token    rgToken[MAX_TOKEN];  /* Pointers to each KEY
*/
auto     int      iToken;
auto     char *   pszWork;
auto     char *   pszTokens;
auto     char *   psz;
auto     int      cb;
auto     RC       rc;

   UNREF(pDBC);

   pszTokens = (char *)ALLOC(STRLEN(pszTokenValues) + 32);
   if (pszTokens == NULL)
      return RDD_NO_MEMORY;

   STRCPY(pszTokens, pszTokenValues);
```
Build the token array.
```
   rc = _dbBuildTokenArrays(pDBC, pszTokens, rgToken);
   if (rc != RDD_SUCCESS)
      {
      FREE(pszTokens);
      return rc;
      }

   juststem(pszSQLSource);

   pszWork = (char *)ALLOC(strlen(pszSQLSource) +
strlen(pszTokenValues) +
             (MAX_TOKEN * strlen(", ")));
   if (pszWork == NULL)
      {
      FREE(pszTokens);
      return RDD_NO_MEMORY;
      }
```
Begin building the EXECUTE SQL statement.
```
   cb = sprintf(pszWork, "EXECUTE %s ", pszSQLSource);

   psz = &pszWork[cb];
```

Add each token/value pair as parameters to the EXECUTE statement.

```
    for (iToken = 0; iToken < MAX_TOKEN; ++iToken)
       {
       if (rgToken[iToken].pszToken == NULL)
          break;

       psz += sprintf(psz, "@%s=%s, ", rgToken[iToken].pszToken,
                  rgToken[iToken].pszValue);
       }
    psz[-2] = '\0';

    *pcbExpandedSQL = strlen(pszWork);

    if (pszExpandedSQL)
       STRCPY(pszExpandedSQL, pszWork);

    FREE(pszWork);
    FREE(pszTokens);

    return RDD_SUCCESS;
}
```

This is the public API called by user applications. This routine has three main sections:

1. Validate all input parameters
2. Based on the fSource parameter, get the SQL statement.
3. Process the SQL statement and apply tokens.

```
/*----------------------------------------------------------------------
--------
Name: rddExpandTokenizedSQL

   Expand a tokenized SQL statement and substitute the supplied
values
----------------------------------------------------------------------
------*/
RDDAPI(RC)          rddExpandTokenizedSQL(
   IN     DBC *     pDBC,
   IN     long      fSource,
   IN     char *    pszSourceSQL,
   OUT    char *    pszExpandedSQL,
   OUT    long *    pcbExpandedSQL,
   IN     char *    pszTokenValues
   )
{
auto      RC        rc;
auto      int       fValidSource = fSource & 0x0F;
auto      BOOL      bLoadedSQL   = FALSE;
auto      char *    pszAllTokens = NULL;
auto      char *    pszSQL;
auto      char *    pszDFT;
auto      char *    pszCMT;
```

Validate the input parameters.

```
   if (pDBC)
      ValidateDBC(pDBC);

   if (pszSourceSQL == NULL)
      return RDD_BAD_ARGUMENT_3;

   if (pszExpandedSQL != NULL)
      {
      if (!_bValidWriteMemory(pszExpandedSQL, *pcbExpandedSQL))
         return RDD_BAD_ARGUMENT_4;
      }

   if (!_bValidWriteMemory(pcbExpandedSQL, sizeof(pcbExpandedSQL)))
      return RDD_BAD_ARGUMENT_5;

   if (pszTokenValues == NULL)
      return RDD_BAD_ARGUMENT_6;
```

Get the SQL statement from one of the various sources. If the source is a stored procedure then the procedure is called directly and control returns to the caller. Each source type has its own read logic except when the SQL statement is passed in as a parameter (fValidSource == SQL_SOURCE_PARM)

```
/* Get the SQL source and Default token key-value pairs
*/
   switch (fValidSource)
      {
      case SQL_SOURCE_FILE:
         if ((rc = _dbReadFileSQL(&pszSourceSQL)) != RDD_SUCCESS)
            return rc;

         bLoadedSQL = TRUE;
         break;

      case SQL_SOURCE_TABLE:
         if ((rc = _dbReadTableSQL(pDBC, &pszSourceSQL)) !=
RDD_SUCCESS)
            return rc;

         bLoadedSQL = TRUE;
         break;

      case SQL_SOURCE_PARM:
         break;

      case SQL_SOURCE_PROC:
         return _dbFormatProcedureCall(pDBC, pszSourceSQL,
                     pszExpandedSQL, pcbExpandedSQL, pszTokenValues
                     );

      default:
         return RDD_BAD_ARGUMENT_2;
      }
```

```
/* -----------------------------------------------------------------
--------
   We have everything we need. Now process the source.

   1. Find and strip the SQL section header.
   2. Find and null all comment sections
   3. Find Default Tokens and concatenate with user passed tokens
   4. Expand the tokenized SQL
   -----------------------------------------------------------------
------*/

   if ((pszSQL = dstrstri(pszSourceSQL, SECTION_SQL)) != NULL)
      --pszSQL;

   pszDFT = dstrstri(pszSourceSQL, SECTION_TOKENS);
```

Strip all the comment sections from the source.

```
   while ((pszCMT = dstrstri(pszSourceSQL, SECTION_COMMENTS)) !=
NULL)
      {
      if (pszCMT < pszSQL)
         MEMSET(pszCMT - 1, ' ', pszSQL - (pszCMT + 1));
      else
      if (pszCMT < pszDFT)
         MEMSET(pszCMT - 1, ' ', pszDFT - (pszCMT + 1));
      else
         pszCMT[-1] = '\0';
      }

   if (pszSQL != NULL)
      MEMSET(pszSourceSQL, ' ', (pszSQL+5) - pszSourceSQL);
```

Source files can have default tokens defined within the source. Build an array with tokens passed in at call time in front of the default token definitions.

```
   pszAllTokens = _dbGetDefaultTokens(pszSourceSQL, pszTokenValues);
   if (pszAllTokens)
      pszTokenValues = pszAllTokens;
```

Finally expand the SQL portion of the source substituting token values for every token found.

```
   rc = _dbExpandTokens(pDBC, pszSourceSQL,
            pszExpandedSQL, pcbExpandedSQL, pszTokenValues
            );
```

If the source was a file or column in a table then a buffer was allocated to hold the source and needs to be freed before exiting.

```
   if (bLoadedSQL)
      FREE(pszSourceSQL);

   if (pszAllTokens)
      FREE(pszAllTokens);

   return rc;
}
```

The foregoing description, for purposes of explanation, used specific nomenclature to provide a thorough understanding of the invention. However, it will be apparent to one skilled in the art that the specific details are not required in order to practice the invention. The foregoing descriptions of specific embodiments of the present invention are presented for purpose of illustration and description. They are not intended to be exhaustive or to limit the invention to the precise forms disclosed. Obviously many modifications and variations are possible in view of the above teachings. The embodiments are shown and described in order to best explain the principles of the invention and its practical applications, to thereby enable others skilled in the art to best utilize the invention and various embodiments with various modifications as are suited to the particular use contemplated. It is intended that the scope of the invention be defined by the following claims and their equivalents: